Recent Work on Reversible Process Calculi

ICCE 2020

Iain Phillips

5 September 2020

Imperial College London

Outline

Reversible computation

Reversible computation allows computation to proceed not only in the standard, forward direction, but also backwards, recovering past states.

Possible applications:

- low-power computing (Landauer 1961)
  if an operation is reversible (e.g. logical negation) then in principle it need not consume energy
- optimistic parallel simulation (Carothers et al 1999)
  reverse computation can be faster than traditional state-saving when rolling back
- error recovery in robot assembly operations (Laursen et al 2015)
- debugging (McNellis et al 2017, Lanese et al 2018)
  record execution, replay forwards and backwards

Outline

Axiomatic approach

Joint work with Ivan Lanese (Bologna) and Irek Ulidowski (Leicester).

FoSSaCS 2020

RCCS

RCCS (Danos & Krivine 2004) is a reversible form of the Calculus of Communicating Systems (CCS) (Milner 1980).

Adds memories to store what has happened already, so that computations can be reversed.

Actions

Example: robot car assembly

- a add left front door
- b add right front door

- $\underline{a}$ remove left front door
- $\underline{b}$ remove right front door

There is no necessary ordering between a and b - different robots?

- c add undercoat of paint
- d add topcoat of paint

- $\underline{c}$ remove undercoat of paint
- $\underline{d}$ remove topcoat of paint

There is an ordering between c and d - we might say that c causes d.

Transitions

We suppress the syntax and just show the labelled transition system:

$$\text{state/process} \xrightarrow{\text{action/label}} \text{new state/process}$$

- Forward transition: $P \xrightarrow{a} Q$
- Reverse transition: $Q \xrightarrow{\underline{a}} P$ undoes $P \xrightarrow{a} Q$
- General (forward/reverse) transition: $t : P \xrightarrow{\alpha} Q$ (defines t)
- Inverse transition: $\underline{t} : Q \xrightarrow{\underline{\alpha}} P$

Can reverse along the same path taken forwards (backtracking):

$$P \xrightarrow{a} Q \xrightarrow{b} R \qquad\qquad R \xrightarrow{b} Q \xrightarrow{a} P$$

But should also take concurrency into account.

Key idea

An action can be reversed if and only if all its consequences have been reversed.

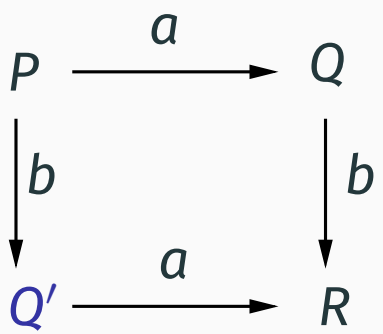
You can take your socks off if and only if you are not wearing shoes.

RCCS

If $P \xrightarrow{c} Q$ causes $Q \xrightarrow{d} R$ then cannot reverse c before d.

But if $P \xrightarrow{a} Q$ and $Q \xrightarrow{b} R$ are independent (concurrent) we can have

$$P \xrightarrow{a} Q \xrightarrow{b} R \qquad\qquad R \xrightarrow{a} Q' \xrightarrow{b} P$$

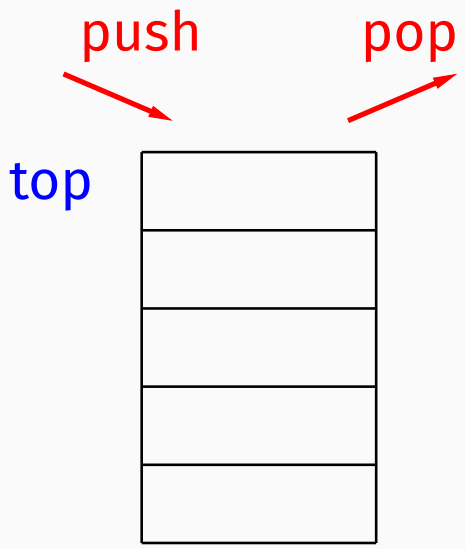


Here Q' was not visited going forwards, but could have been:

$$P \xrightarrow{b} Q' \xrightarrow{a} R \qquad\qquad R \xrightarrow{a} Q' \xrightarrow{b} P$$

Memories

RCCS stores information needed to reverse a computation in memories, which are stacks onto which new information is pushed for each new transition.
This information is popped off the stack when the corresponding transition is reversed.
There are different stacks for the different parallel threads.



Concretely, independence means that memories are *coherent* (non-overlapping), so that the transitions belong to different threads.

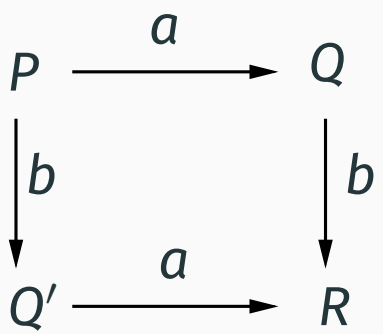
Causal Equivalence

Causation is not modelled directly in RCCS.

Paths r, s are sequences of transitions $t_1 t_2 \ldots t_n$.

Causal equivalence on paths:
$r \approx s$ if and only if s can be got from r by

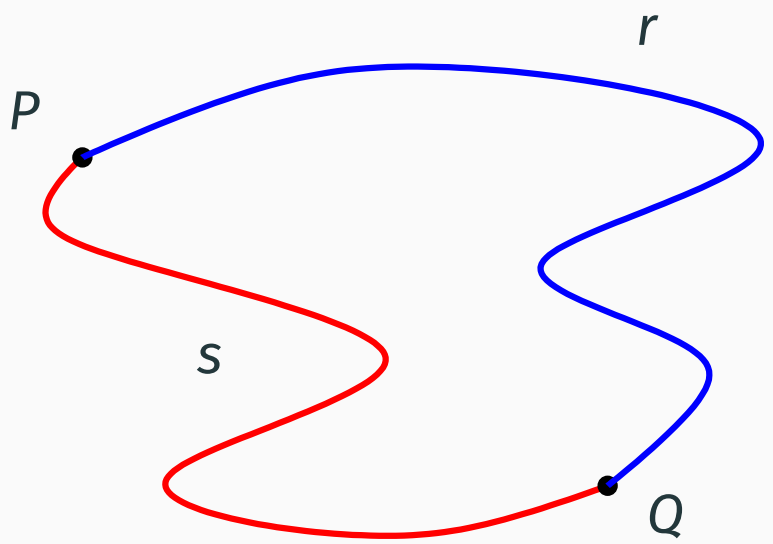
1. swapping adjacent independent transitions
2. cancellation $t\underline{t} = \underline{t}t = \epsilon$



Causal Consistency

Causal Consistency (CC)

If r and s are coinitial and cofinal paths then $r \approx s$.



Causal Consistency

CC has become the dominant property shown for reversible languages.

CC has been shown for RCCS and numerous other reversible calculi:

- μOz (Lienhardt et al 2012)
- Klaim (tuple-based language, Giachino et al 2017)
- reversible Erlang (Lanese et al 2018)
- reversible occurrence (Petri) nets (Melgratti et al 2019)

Proofs are quite lengthy but mostly take a similar approach.

Our approach

Our idea

If we can show that properties such as CC follow from a small set of axioms, it should be easier to check these properties by verifying the axioms.

We use abstract labelled transition systems with independence (LTSIs).

Related to the occurrence LTSIs of Sassone et al (1996).

- We adopt a minimal set of axioms and add more as needed
- We treat reverse transitions as first-class citizens

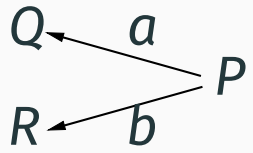
Our approach

Usual proof of CC involved showing the Parabolic Lemma (PL) and then showing a weaker form of CC by various means depending on the context.

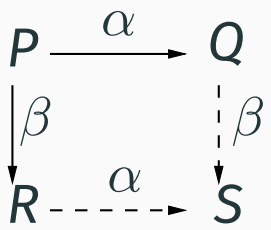
By studying basic axioms we have the following:

Axioms

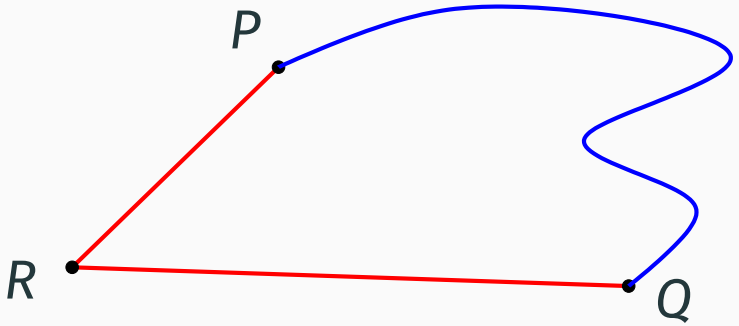
1. If backward transitions are coinitial then independent:



2. Square property: If transitions are coinitial and independent then:



From these can deduce Parabolic Lemma:



Axioms

3. Well-foundedness (WF): no infinite reverse path

$$\cdots \xrightarrow{a_{n+1}} P_n \xrightarrow{a_n} \cdots \xrightarrow{a_2} P_1 \xrightarrow{a_1} P_0$$

Satisfied by all reversible calculi we know of.

Cannot reverse to before starting point.

Theorem

If WF and PL then CC.

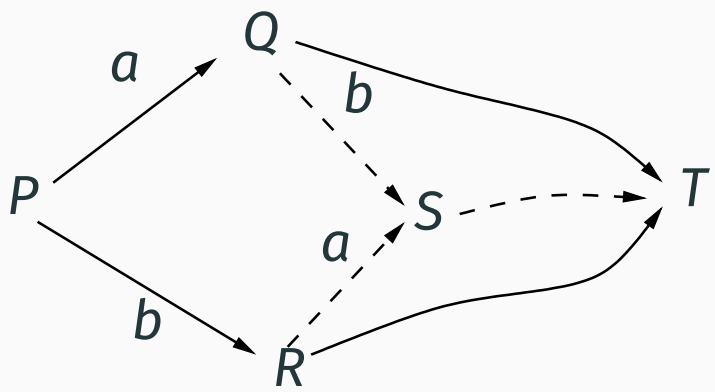
- Success for the axiomatic approach
- Much shorter than existing proofs
- Shows that CC is not really stronger than PL

CC

Existing proofs of CC do not appear to use WF.

They do use further properties which are not deducible from our axioms.

Forward confluence:



Beyond CC

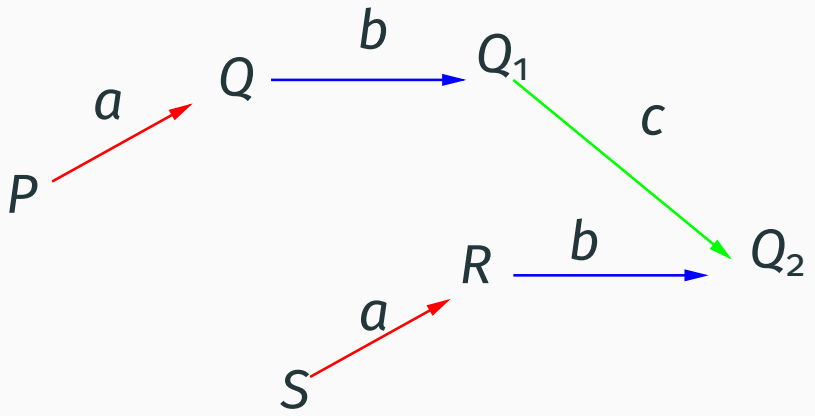
If CC is weaker than thought, how should we characterise our Key Idea?

Split into:

- Causal Safety: if can reverse *t* then all its consequences have been undone
- Causal Liveness: if all *t*'s consequences have been undone then can reverse *t*

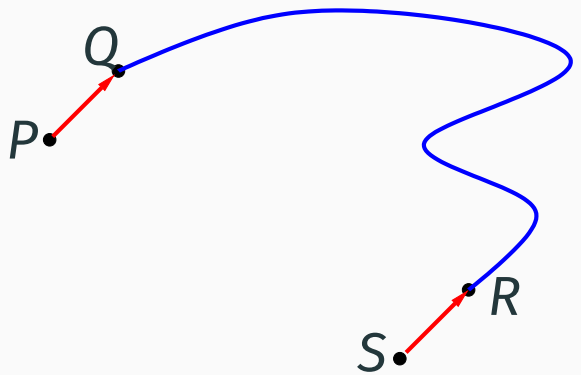
Causal Safety

Causal Safety (CS):



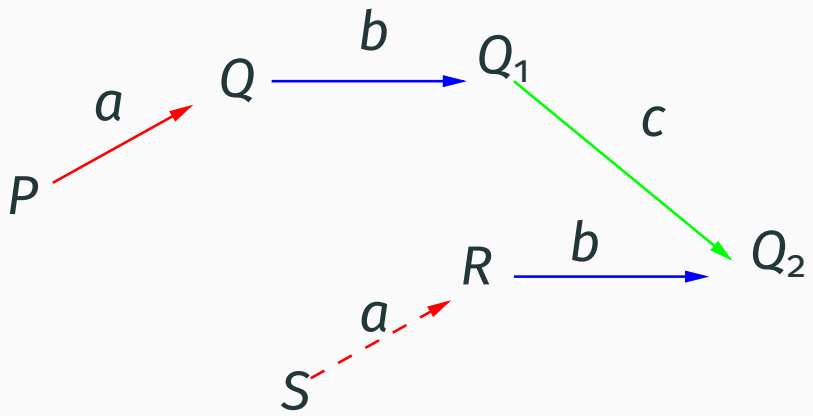
Here b occurs net zero times; c is net positive and must not be caused by a.

More generally:



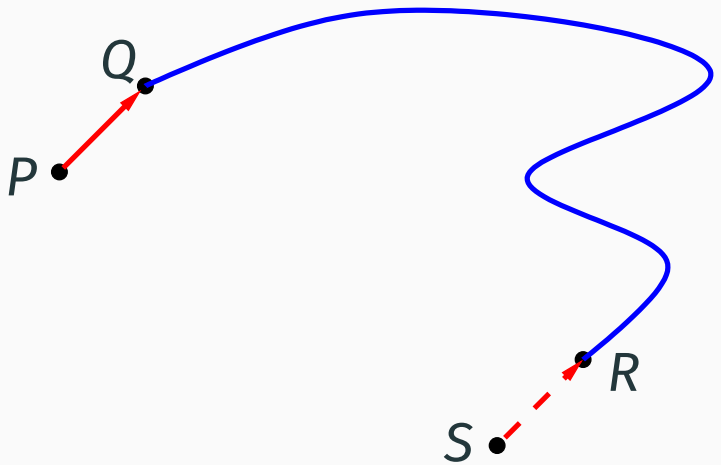
Causal Liveness

Causal Liveness (CL):



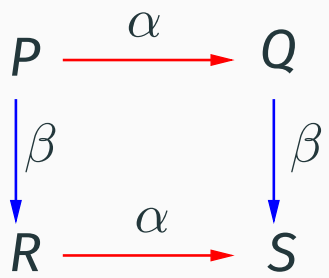
If c is not caused by a then a can reverse at R.

More generally:



Events

Since transitions $P \xrightarrow{a} Q$ may be reversed later by $S \xrightarrow{a} R$ we need to group together transitions into events.



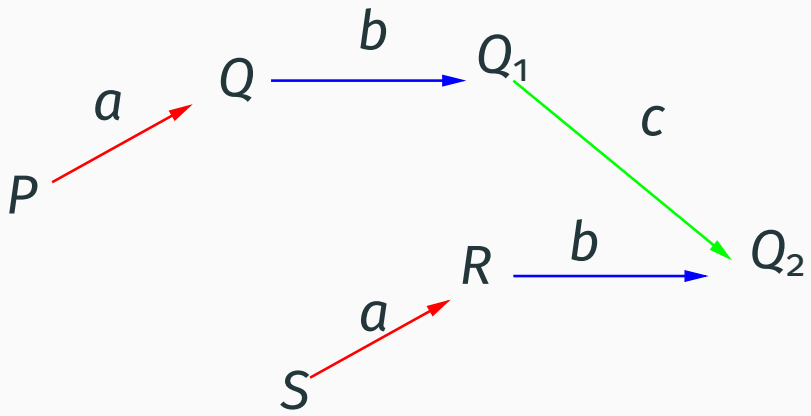
If coinitial transitions in the square are independent then we let

$$P \xrightarrow{\alpha} Q \sim R \xrightarrow{\alpha} S \qquad P \xrightarrow{\beta} R \sim Q \xrightarrow{\beta} S$$

Get two events $[P \xrightarrow{\alpha} Q]$ and $[P \xrightarrow{\beta} R]$.

Lift independence to events: $[t_1]$ ci $[t_2]$ if have representatives t_1' and t_2' which are coinitial and independent.

Causal Safety (CS)



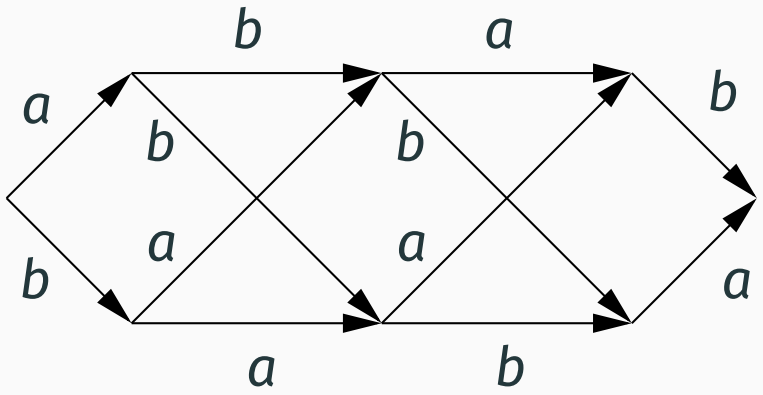
We give three definitions of causal safety:

1. via independence of transitions ($P \xrightarrow{a} Q \iota Q_1 \xrightarrow{c} Q_2$)
2. via ordering of events ($[P \xrightarrow{a} Q] \not< [Q_1 \xrightarrow{c} Q_2]$)
3. via independence of events ($[P \xrightarrow{a} Q]$ ci $[Q_1 \xrightarrow{c} Q_2]$)

With minimal axioms these are all different, but with our full set of axioms they become equivalent.

Example

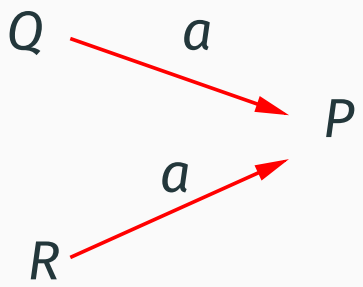
Satisfies all axioms so far and therefore PL+CC, but not CS and not CL:



We provide further axioms from which CS and CL can be deduced.

Reverse event determinism

RED: reverse event determinism



If $t_1 : Q \xrightarrow{a} P \sim t_2 : R \xrightarrow{a} P$ then $t_1 = t_2$.

A natural property for reversible systems.

Theorem

The following are equivalent under minimal axioms:

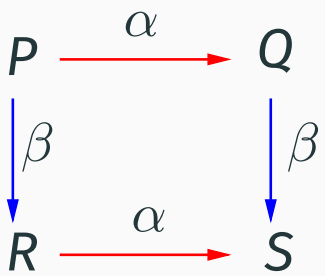
1. *NRE: no event can occur (net) twice in a path*
2. *RED: reverse event determinism*
3. *Polychotomy: events are ordered, in conflict or independent*

Showing CC is not enough to show RED - need further axioms.

Axioms

1. SP: square property
2. BTI: backward transitions are independent
3. WF: well-founded

4. CPI: coinitial propagation of independence (around a square)



5. IRE: independence respects events (if $t \sim t' \iota u$ then $t \iota u$)
6. IEC: independence of events is coinitial (if $t \iota u$ then $[t]$ ci $[u]$)

Case Studies

All our axioms hold in the following settings:

- RCCS
- a labelled version of reversible higher-order π-calculus HOπ (Lanese et al 2016)
- a labelled version of reversible Erlang (Lanese et al 2018)
- reversible occurrence (Petri) nets (Melgratti et al 2019)

So in each case we deduce CC, CS and CL from our general abstract results.

Our work does not apply as it stands to the reversible π-calculus Rπ (Cristescu et al 2013), since transitions in the same event have equivalent rather than identical labels.

Reversible systems

A longer-term goal would be to characterise reversible systems using axioms.

We have given a fairly minimal set of fundamental axioms aimed at being sufficient to yield the Key Idea.

This does not include forward confluence, even though this does hold for RCCS.

Previous work showed that a different set of axioms (including forward confluence) characterises a class of reversible transition systems which correspond to prime event structures with non-binary conflict relation (Phillips and Ulidowski 2007).

Non-binary conflict: e.g. at most two of three events a, b, c can occur.

RCCS looks like it has a binary conflict relation $a \# b$.

Connections with the occurrence LTSIs of Sassone et al (1996).

Summary

- We present basic axioms which are satisfied by RCCS and other reversible calculi.
- For a new reversible languages, verifying these axioms will be easier than verifying less basic properties.
- We have seen that Causal Consistency is not sufficient, and should be supplemented by Causal Safety and Causal Liveness.
- Our abstract proofs are relatively easy to formalise in a proof assistant (interactive theorem prover software).

Outline

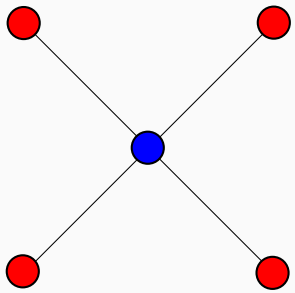
Retrospection

Joint work with Eva Graversen (Imperial), Jean Krivine (Paris) and Nobuko Yoshida (Imperial).

Motivation

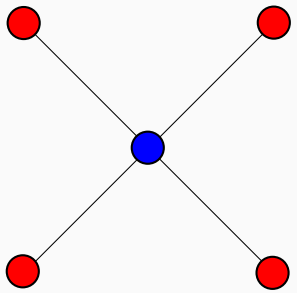
Processes need to reach consensus on when to commit to a transaction.

For instance, different databases need to be updated in a consistent way.



Possibilties of failure at nodes or links.

Two-phase commit protocol



A simple two-phase commit protocol

Coordinator sends query to participants.
Participants either prepare for commit or they abort.
Participants send vote yes/no to coordinator.
If any no vote received (or time out) coordinator sends rollback message to participants.
If all votes are yes then coordinator sends commit message to participants.
Participants either rollback or commit locally and send acknowledgement to coordinator.

Idea

Treat distributed consensus in a declarative manner:

- task of aborting and committing transactions, and notifications of abort/commit delegated to the operational semantics.

Processes can be in three states of belief about the transaction they participate in:

- proceed
- fail
- success

Transactions

Transaction:

- if succeed then the leader commits and others need to learn this
- if a participant fails (e.g. times out) then roll back including related actions by other participants

New calculus

RCCS is uncontrolled reversibility (reverse anything if consequences have been reversed)

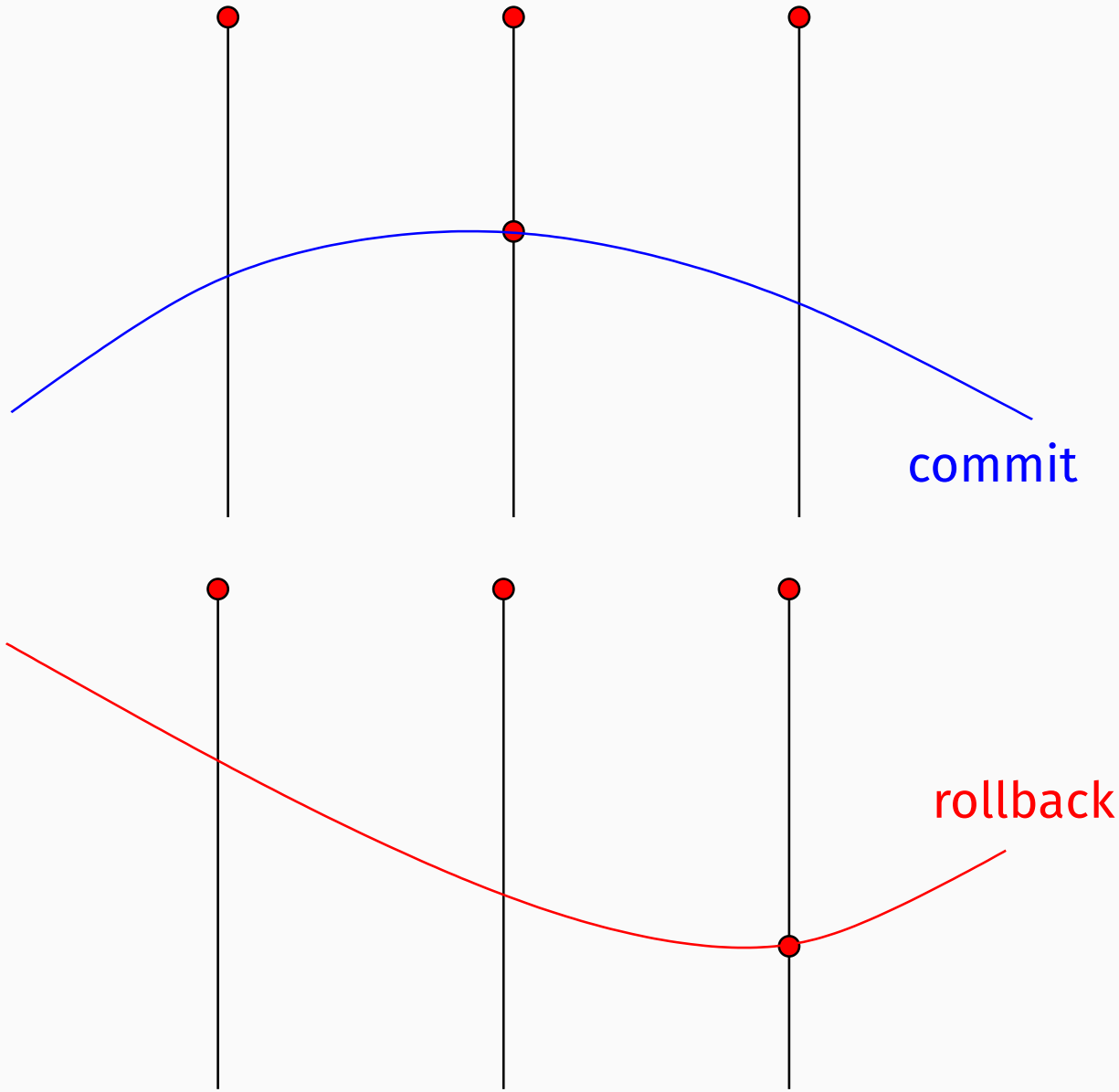
Instead, allow rollbacks to labelled checkpoints.

This means using memories to reverse, and removing items from memory as we reverse.

Commits are irreversible.

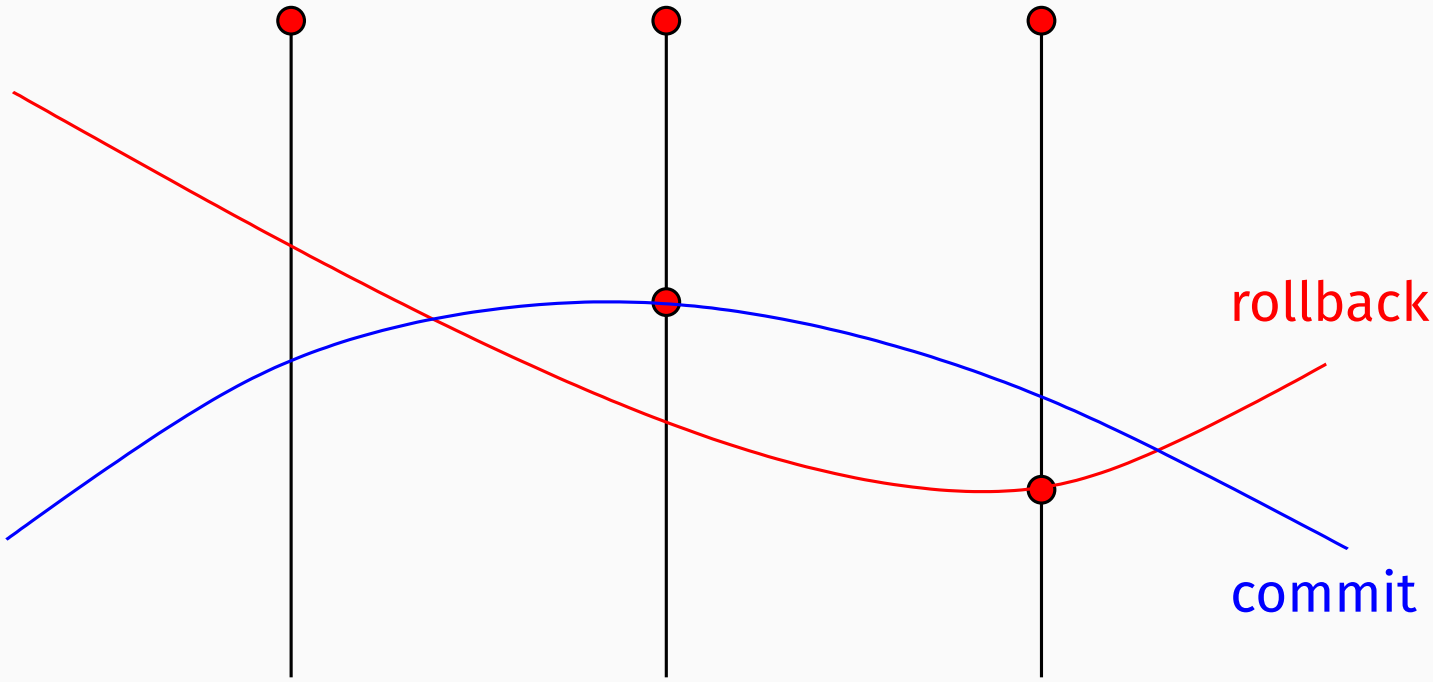
Rollbacks have previously been used in other process calculi (Lanese et al 2011).

Duality of Commits and Rollbacks



Design Decision

Commits take priority over rollbacks if they conflict.



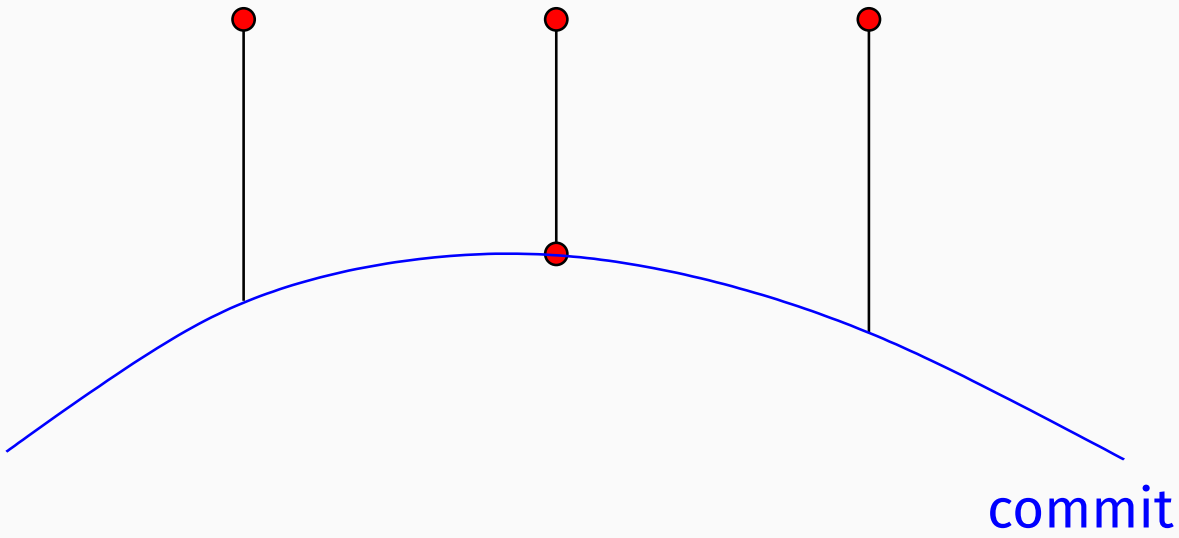
Notice that we allow commitment some time after the transaction is completed.

Subtleties if have multiple commits and rollbacks: resolved by marking memories prior to deletion

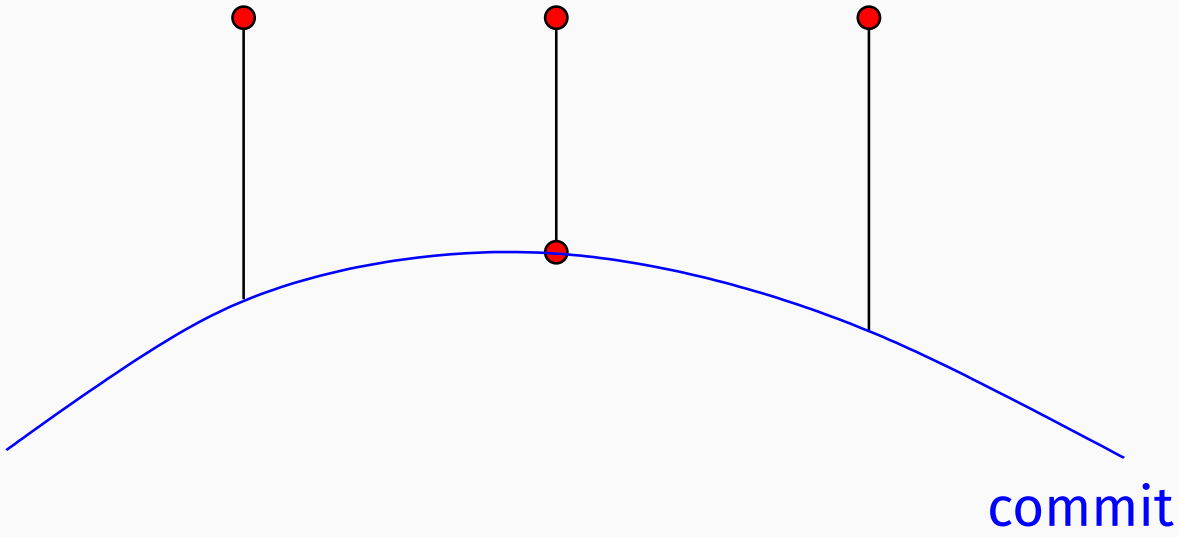
Retrospection

New idea: retrospection

In most reversible calculi, memories are not deleted when commit.
We want to garbage collect.



Retrospection



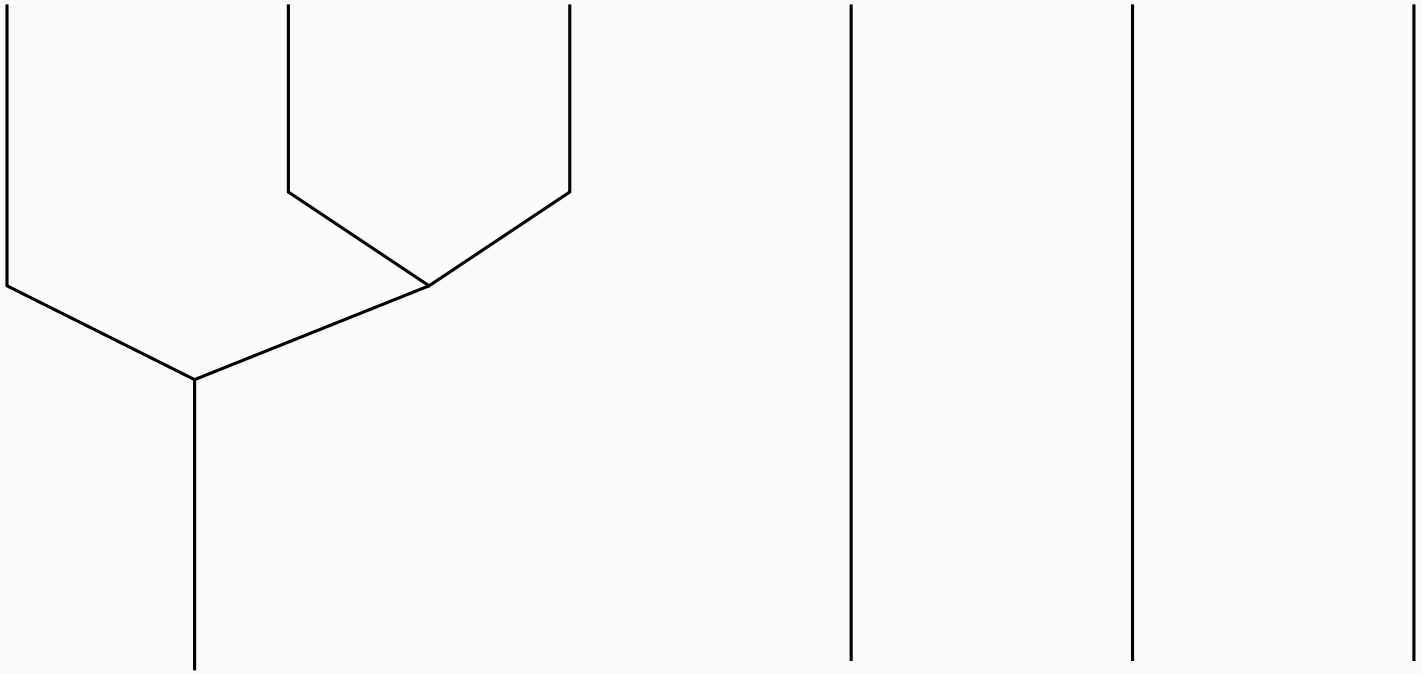
Different threads share a conceptual memory which is distributed over the threads. If a commit occurs on one thread, this leads to garbage collection of memories across all threads.

Then the other threads can inspect their memories and deduce that a commit occurred remotely.

They can then continue with other activities.

Process trees and coherence

Process trees are the high-level specification implemented by threads with different memory stacks (local copies).



Branching occurs when threads split and execute concurrently, keeping their memories coherent.

Process trees and coherence

Pre-coherent defined inductively on syntax:

- threads split in a consistent manner
- no causal cycles

Coherent:

- pre-coherent
- complementary memory keys corresponding to opposite sides of synchronous communication

Then:

- reachable by open transitions implies pre-coherent
- reachable by closed transitions implies coherent

Confluence

We show that the process of marking memories for removal and removing them is confluent (the order does not matter), and corresponds to the expected high-level specification in terms of process forests.

Summary

- The new version of RCCS has explicit commits and rollbacks
- It can combine multiple commits and rollbacks in a consistent fashion
- It can model protocols such as two-phase commit
- Since the notification of success is delegated to the infrastructure via retrospection, costs of implementation are alleviated, at least as far as the programmer is concerned.

Outline

Event Structure Semantics

Joint work with Eva Graversen and Nobuko Yoshida.

RC 2018, RC 2020

Motivation

We have seen events as equivalence classes of transitions

$$[t : P \xrightarrow{a} Q]$$

These events $a, b, \ldots$ are related in three ways:

1. Event a causes event b
2. a and b are in conflict — they cannot both occur
3. a and b are concurrent — neither of the above

We have seen two types of reversibility:

- uncontrolled: events can be reversed if all their consequences have been reversed
- controlled: events can only be reversed using explicit rollbacks

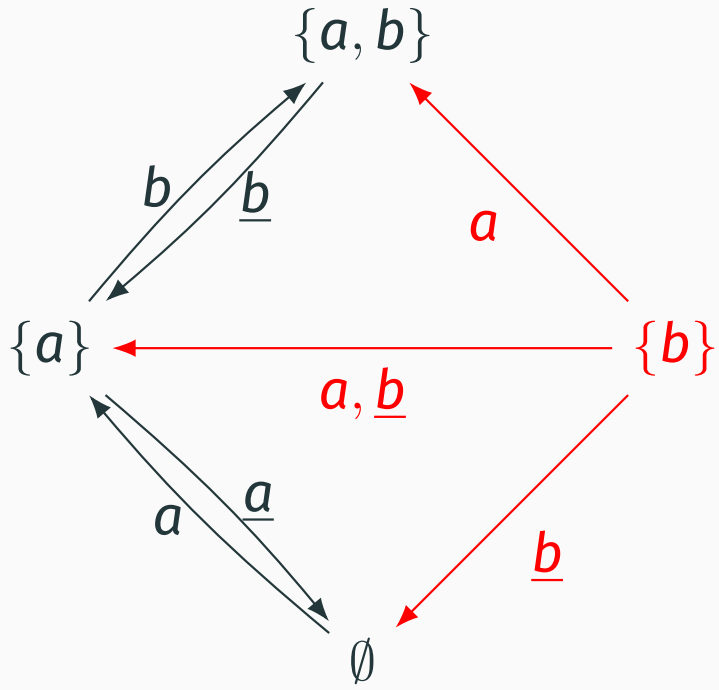
Reversible event structures

We use two different types of reversible event structure to model these two possibilities.

Reversible event structures were first defined in Phillips and Ulidowski (2013) and Phillips, Ulidowski and Yuen (2013).

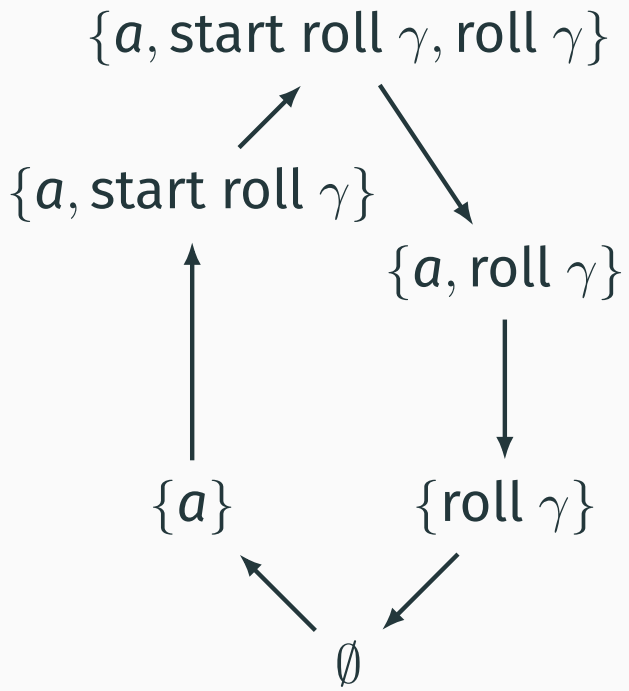
Uncontrolled reversibility

Events a, b
Reversible events a, b
Causation $\{a\} \mapsto b$,
Prevention $b \rhd \underline{a}$



Controlled reversibility

$a_\gamma.\text{roll}\ \gamma$



Summary

- We model reversible CCS (uncontrolled) using causal reversible event structures
- We model CCS with rollbacks (controlled) using non-causal reversible event structures
- We have further modelled the reversible internal π-calculus using causal event structures
- The longer term aim is to contribute to the discussion on the open question of when two reversible processes should be considered to be equivalent.